Name: Sahib Bajwa

ID: 107553096

CSCI 3104, Algorithms **Profs. Chen & Grochow**

Problem Set 5 – Due Thurs Feb 20 11:55pm **Spring 2020, CU-Boulder**

Advice 1: For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.

Advice 2: Informal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.

Instructions for submitting your solutions:

- All submissions must be easily legible.
- You should submit your work through the **class Canvas page** only.
- You may not need a full page for your solutions; pagebreaks are there to help Gradescope automatically find where each problem is. Even if you do not attempt every problem, please allot at least as many pages per problem (or subproblem) as are allotted in this template.

Quicklinks: 1a 1b 2a 2b 3a 3b 3c

---

1. *In this question we consider the change-making problem (as covered in recitation), of making change for n cents using the smallest number of coins. Suppose we have coins with denominations of $v_1 > v_2 > \cdots > v_r$ for r coins types, where each coin's value v_i is a positive integer. Your goal is to obtain a set of counts $\{d_i\}$, one for each coin type, such that $\sum_{i=1}^{r} d_i = k$ and where k is minimized, and such that the sum of the values $\sum_{i=1}^{r} d_i v_i = n$.*

   (a) *A greedy algorithm for making change is the **cashier's algorithm**. Consider the following pseudocode—meant to implement the cashier's algorithm—where n is the amount of money to make change for and v is a vector of the coin denominations:*

```
change(n,v,r) :
   d[1 .. r] = 1       // initial histogram of coin types in solution
   while n > 0 {
      k = r
      while ( k > 0 and v[k] > n ) { k++ }
      if k==0 { return 'no solution' }
      else { n = n - v[k] }
   }
   return d
```

Name: Sahib Bajwa
ID: 107553096

CSCI 3104, Algorithms **Profs. Chen & Grochow**
Problem Set 5 – Due Thurs Feb 20 11:55pm **Spring 2020, CU-Boulder**

This code has bugs. Identify the bugs and explain why each would cause the algorithm to fail.

1. You are incrementing k by +1 so the while loop will never exit. Decrement k instead: k = k - 1
2. You are not manipulating d at all, so you will return the initially declared array. In the else statement should be: d[k] = d[k] + 1
3. Indexing may be off? Depends on how the indexing for the pseudocode was done. Here is a picture for how I solved indexing the code in python:

```
def change(n, v, r):
    k = 0
    d = [0] * r
    while(n > 0):
        k = r
        while(k > 0 and v[k-1] > n):
            k = k - 1
        if(k == 0):
            return("no solution")
        else:
            d[k-1] = d[k-1] +1
            n = n - v[k-1]
    return d
```

(b) *Identify a set of Euro coin denominations (a subset of the denominations here)*[1] *for which the greedy algorithm does not yield an optimal solution for making change. Justify your answer in terms of optimal substructure and the greedy-choice property. (The set should include the 1 Euro cent so that there is a solution for every value of n.) Include an example where the cashier's algorithm with your choice of denominations yields a set of coins that is larger than it needs to be, and also show the smaller set of coins adding up to the same value.*

Coinset = [1, 20, 50] is the set of Euro coin denominations I am going to be using. For the value 60, the optimal substructure property fails. The optimal substructure property fails because the optimal solution for the entire problem does not use the optimal solution for each sub-problem. Some of the choices made for the sub-problems are not the optimal solution for the sub-problem, but creates an optimal solution for the entire problem. The greedy choice property still holds because you can come to a solution incrementally without reference to future decisions due to us including 1 in the set. Example: For a coin amount of 60, the greedy algorithm tells us to use one 50 coin denomination and ten 1 coin denominations for a total of 11 coins used. The optimal solution however is to use three 20 coin denominations for a total of 3 coins used.

[1] `https://www.google.com/search?q=euro+coin+denominations`

Name: Sahib Bajwa
ID: 107553096

CSCI 3104, Algorithms **Profs. Chen & Grochow**
Problem Set 5 – Due Thurs Feb 20 11:55pm **Spring 2020, CU-Boulder**

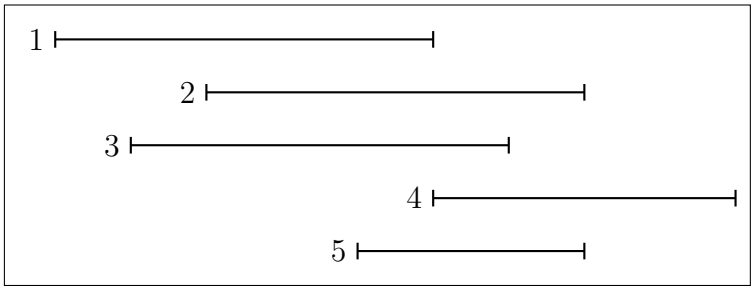
2. *In this question we consider the interval scheduling problem, as covered in class.*

   (a) *Consider a greedy algorithm which always selects the shortest appointment first. Draw an example with at least 5 appointments where this algorithm fails. List the order in which the algorithm selects the intervals, and also write down a larger subset of non-overlapping intervals than the subset output by the greedy algorithm.*

   A comment on level of justification (applies to all of problems 2 and 3): *to help us understand your thinking, it is worth writing a little about the order in which the algorithm selects the intervals. For example "The algorithm takes intervals in the order [1,3,4]: first the algorithm takes interval 1 because that is the shortest. Interval 1 conflicts with intervals 2 and 5, so they are removed. The next shortest is 3, which conflicts with interval 6, and the only remaining interval is 4."*

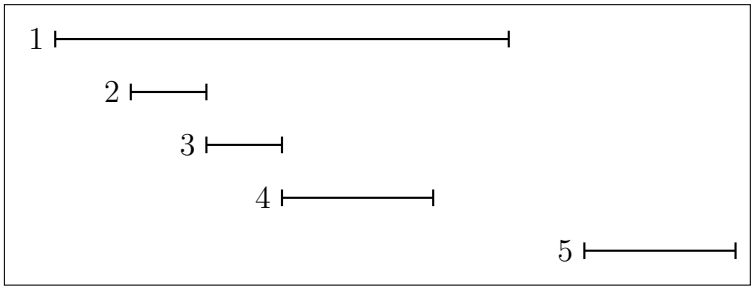
   YOUR ANSWER HERE. We've provided you with some sample code to draw such a figure natively in LaTeX, you just need to modify the start/end points and possibly add more intervals if you need (as well as answer the rest of the question!).

   The algorithm takes intervals in the order [5]: first the algorithm takes interval 5 because that is the shortest interval. Interval 5 conflicts with all other intervals, so they are removed. Since all other intervals are removed, [5] is the entire order of intervals for the algorithm. The best scheduling would have been intervals in the order [1,4]: they are the only two elements that do not overlap and coincidentally cover the most amount of time overall.



(b) *Consider a greedy algorithm which always selects the longest appointment first. Draw an example with at least 5 appointments where this algorithm fails. Show the order in which the algorithm selects the intervals, and also show a larger subset of non-overlapping intervals than the subset output by the greedy algorithm. The same comments apply here as for 2a in terms of level of explanation.*

The algorithm takes intervals in the order [1,5]: first the algorithm takes interval 1 because that is the longest interval. Interval 1 conflicts with intervals 2, 3, and 4, so they are removed. The algorithm then takes interval 5 as it is the next longest interval. No other intervals can be scheduled, so the order of intervals for the algorithm is [1,5]. The best scheduling would have been intervals in the order [2,3,4,5]: this covers the most intervals possible and intervals 2,3, and 4 are in conflict with 1, so interval 1 cannot be scheduled.

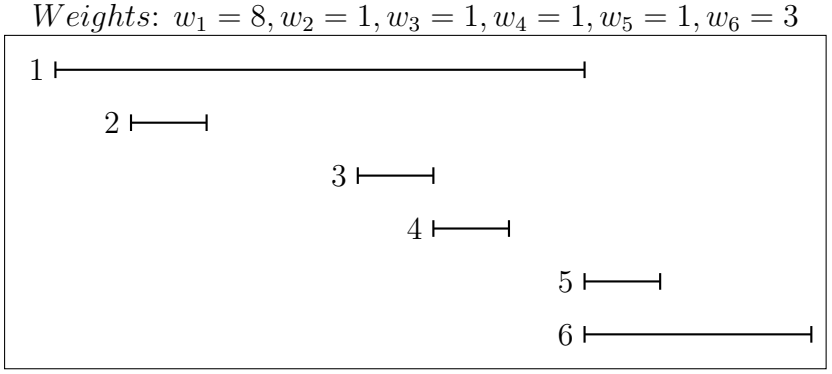


Name: Sahib Bajwa
ID: 107553096

3. *In this question we'll consider weighted problems.*

   (a) *Consider the* weighted *interval scheduling problem. In this problem, the input is a list of n intervals-with-weights, each of which is specified by* $(start_i, end_i, wt_i)$*. The goal is now to find a subset of the given intervals in which no two overlap and to maximize* the sum of the weights*, rather than the total number of intervals in your subset. That is, if your list has length n, the goal is to find* $S \subseteq \{1, \ldots, n\}$ *such that for any* $i, j \in S$*, interval i and interval j do not overlap, and maximizing* $\sum_{i \in S} wt_i$*. Consider the greedy algorithm for interval scheduling from class, which selects the job with the earliest end time first. Give an example of weighted interval scheduling with at least 5 intervals where this greedy algorithm fails. Show the order in which the algorithm selects the intervals, and also show a higher-weight subset of non-overlapping intervals than the subset output by the greedy algorithm. Same comments apply as on problem 2.*

   The greedy algorithm takes intervals in the order [2,3,4,5]: 2, 3, and 4 are the earliest 3 intervals to finish and do not conflict with each other. All 3 intervals (2, 3, and 4) conflict with interval 1, so it cannot be scheduled. the next scheduled interval is 5, which conflicts only with 6, so it cannot be scheduled. The total weight of [2,3,4,5] is 4. The highest weight scheduling is [1,6], which has a weight of 11.



(b) *Consider the Knapsack problem: the input is a list of n items, each with a value and weight* (val_i, wt_i)*, and a threshold weight* W*. All values and weights are strictly positive. The goal is to select a subset* S *of the items such that* $\sum_{i \in S} wt_i \leq W$ *and maximizing* $\sum_{i \in S} val_i$*. (Note that,* unlike *change-making, here there is only one of each item, whereas in change-making you in principle have an unlimited number of each type of coin.) Consider a greedy algorithm for this problem which makes greedy choices as follows: among the remaining items, choose the item of maximum value such that it will not make the total weight exceed the threshold* W*. Give an example of knapsack with at least 5 items where this greedy algorithm fails. Show the order in which the algorithm selects the items, and also show a higher-value subset whose weight does not exceed the threshold. Same comments apply as on problem 2.*

Threshold weight (W) = 10

Item Number	Value	Weight
1	1	1
2	6	2
3	4	5
4	5	11
5	7	9

The algorithm first takes item 5 to put in the knapsack. This is because 5 has the largest value at 7. Since the weight of item 5 is 9 and our threshold weight (W) is 10, we have to exclude items 2-4 due to their weights putting us over the threshold. The only item left and the one that will not put us over the threshold weight is item 1, so we put item 1 in the knapsack. The ending value for all the items in the knapsack is 8 and the overall weight is 10.

In the case of these items, the best way to fill the knapsack would be to take item 2 first. Taking item 2 first will exclude items 4 and 5. Then taking item 1, which will exclude nothing new. Then taking item 3, which leaves us with no items left. The ending value for all the item sin the knapsack is 11 and the overall weight is 8.

(c) *Now consider the following algorithm for the knapsack problem. Call the* relative value *of item i the ratio val_i/wt_i. Consider the greedy algorithm which, among the remaining items, chooses the item of maximum* relative value *such that it will not make the total weight exceed the threshold W. Give an example of knapsack where this greedy algorithm fails. Show the order in which the algorithm selects the items, and also show a higher-value subset whose weight does not exceed the threshold. Same comments apply as on problem 2.*

Threshold weight (W) = 10

Item Number	Value	Weight
1	7	6
2	5	5
3	5	5
4	1	4
5	7	10

The first value the algorithm takes is item 1 since its $\frac{value}{weight}$ is the highest at $\frac{7}{6}$ (which is 1.167). Since the weight of item 1 is 6 and our threshold weight (W) is 10, we have to exclude items 2, 3, and 5 due to their weights putting us over the threshold. The algorithm then takes item 4 since it is the only item left and it does not cause us to go over the threshold weight. This gives us a final value of 8 and a total weight of 10, with a relative value of $\frac{8}{10}$ (which is 0.8).

The optimal item selection would be to first take item 2. Since item 2 has a weight of 5, we have to exclude items 1 and 5 due to their weights putting us over the threshold. The next item to take would be item 3. Since item 3 has a weight of 5 putting us at a total weight of 10, we have to exclude item 4 due to its inclusion putting us over the threshold weight. This would give us a final value of 10 and a total weight of 10 with a relative value of $\frac{10}{10}$ (which is 1). Not doing this greedy algorithm allows for us to have more value in the knapsack with the same total weight.